dynamicWEB®

User manual

Extensibility API

How to extend Dynamicweb eCommerce

Version: 1.50
2008.11.21
English

www.dynamicweb.dk | Åhavevej 5 | DK - 8260 Viby J | ☎ +45 7025 2090 | +45 7025 2091

LEGAL INFORMATION

© Copyright 2007-2008 Dynamicweb Software A/S (Ltd).

All rights reserved. Alteration or reproduction of this document or parts hereof is strictly prohibited, regardless of form or means, unless explicit permission has been acquired from Dynamicweb Software.

Dynamicweb is a registered trademark of Dynamicweb Software.

Company and product names mentioned in this document may be registered trademarks or trademarks of third parties.

CONTENTS

Introduction _____ vi
What this document is about vi
Who this document is for vi
Related documents vi

1 Introducing the Extensibility API _____ 1
1.1 Providers overview 1
- 1.1.1 Price Providers 1
- 1.1.2 Fee Provider 1
- 1.1.3 Sales discount Providers 1
- 1.1.4 Payment Gateway Providers 1
- 1.1.5 Statistic Providers 1
- 1.1.6 Import/Export Activity Providers 1
- 1.1.7 Notification Subscribers 1
- 1.1.8 Template Extenders 1
- 1.1.9 Menu providers 1

2 Price Providers _____ 2
2.1 Understanding prices 2
2.2 Raw Price 2
2.3 PriceProviders and sales tax 3
2.4 Currency and rounding 3
2.5 Code sample 3

3 Fee Provider _____ 5

4 Sales Discount Providers _____ 6
4.1 Different types of sales discounts 6
4.2 Sales discount types and their parameters 6
4.3 Creating your own sales discount 6
4.4 How are the SalesDiscounts evaluated? 6
4.5 Code sample 6

5 Gateway Providers _____ 11
5.1 Code sample 11

6 Statistics Providers _____ 13
6.1 Delivering data 13
6.2 Extra information 13
6.3 Own XML data format 13
6.4 Change the default graph look 13
6.5 Code sample 13

7 Import/Export Activity Providers _____ 19

© 2007-2008 Dynamicweb Software A/S (Ltd).

7.1 Datatype conditions.....19
7.2 What about reports?.....19
7.3 How do I decide what my activity should do?.....19
7.4 Code sample - Activity.....20

8 Notification Subscriber _____ 23

8.1 Notifications.....23
8.2 Code samples.....23
- 8.2.1 Basic example.....24
- 8.2.2 Product List Modifier.....24
- 8.2.3 Subscribe to module events.....28
- 8.2.4 Extranet login notification.....29
- 8.2.5 Custom order line fields modifier.....29

9 Template Extenders _____ 31

9.1 Extender types.....31
- 9.1.1 ProductTemplateExtender.....31
- 9.1.2 ProductListTemplateExtender.....31
- 9.1.3 OrderTemplateExtender.....31
- 9.1.4 OrderLineTemplateExtender.....31
- 9.1.5 PageTemplateExtender.....31

9.2 Template Extenders RenderingState.....31
9.3 Code samples.....32
- 9.3.1 ProductTemplateExtender.....32
- 9.3.2 ProductListTemplateExtender.....32
- 9.3.3 OrderTemplateExtender.....32
- 9.3.4 OrderLineTemplateExtender.....33
- 9.3.5 PageTemplateExtender.....33

10 Menu provider _____ 34

© 2007-2008 Dynamicweb Software A/S (Ltd).

INTRODUCTION

What this document is about

This document describes the Extensibility API for Dynamicweb eCommerce.

Who this document is for

This document is for developers that want to work with the Dynamicweb eCommerce Extensibility API in order to extend the functionality in Dynamicweb eCommerce.

Related documents

- **Payment Gateways, Dynamicweb eCommerce**: Gives a general description of how to use the *Payment Gateway* module for Dynamicweb eCommerce.
- **Document Conventions**: Gives an overview of the typographical and writing conventions used in manuals and documentation from Dynamicweb Software.

© 2007-2008 Dynamicweb Software A/S (Ltd).

1 INTRODUCING THE EXTENSIBILITY API

1.1 Providers overview

1.1.1 Price Providers

Provides a way to replace and extend the existing price structure of Dynamicweb eCommerce, and to make calls to external systems to get a product price.

1.1.2 Fee Provider

The provider allows you to manipulate shipping fees on orders in Dynamicweb eCommerce.

1.1.3 Sales discount Providers

In short, *SalesDiscountProviders* is to orders, what *PriceProviders* is to products. It provides you with the possibility to invent your own discount types from scratch and apply them to an order, or to extend the built-in discount types in the *Sales Discount* module.

1.1.4 Payment Gateway Providers

Need to support a gateway that is not included in our standard product. No problem. Write your own *PaymentGatewayProvider* by providing the template and a few lines of code (such as a new hashing algorithm), and plug it into Dynamicweb eCommerce right away. It will work side by side with the built in payment gateways in Dynamicweb eCommerce.

1.1.5 Statistic Providers

The provider allows you to present data that is not included in the standard *Statistics, eCom* module.

1.1.6 Import/Export Activity Providers

Lets you create your own activities for import of your own data as well as transformation from your own format.

1.1.7 Notification Subscribers

Allow developers to subscribe to system events, such a new orders, or changes in stock states. Use them to update external systems with order data, or just to send an e-mail to the shop manager when the stock is running low.

1.1.8 Template Extenders

Need a new tag? With *Template Extenders*, developers can hook into the rendering process of a product, a product list, or the cart, and extend the template with new tags or redefine existing tags.

1.1.9 Menu providers

Lets you add your own custom menu items in XSLT menus.

 © 2007-2008 Dynamicweb Software A/S (Ltd).

2 PRICE PROVIDERS

A *PriceProvider* provides Dynamicweb eCommerce with an algorithm for retrieving a specific price for a product. A *PriceProvider* overrides the default algorithm of the system by returning a price.

It is up to the developer of the *PriceProvider* to decide how the *Price* data is retrieved, cached and updated. This feature makes it possible for developers to call external systems and query for price data.

The following rules apply for *PriceProviders*:

- Multiple *PriceProviders* are allowed.
- A *PriceProvider* should never return a *Price* if it does not have one for a particular situation. In the case that a *PriceProvider* cannot resolve a *Price*, the system will call the next *PriceProvider* in the *Provider* chain.
- If no *PriceProvider* exists for a price, the system will fall back to the *DefaultPriceProvider* (Product Price Matrix).

2.1 Understanding prices

A *price* is the cost per unit of a product sold in a specific context. When the system asks a *PriceProvider* for a price, it calls the *PriceProvider* with a set of parameters that together define the context.

As developer of the *PriceProvider*, you can choose to use one or more of the parameters to resolve the price. You can also ignore parameters that are not relevant for resolving a price. For example, if you are not concerned with the quantity parameter (because the price is not dependant of quantity), you can choose to overlook this parameter by returning the same price no matter what quantity the system asks for.

The context of the price is determined by the following parameters:

- **Product**
  The product that the system needs to resolve a price for.
- **Quantity**
  The number of product items in a given unit type (see below) that the system needs to resolve a price for.
- **VariantID**
  The ID of a specific product variant that the system needs to resolve a price for.
- **Currency**
  The currency of the requested price. Only return a price in this currency. If no price exists for this currency then return null (Nothing) instead. The provider will be asked again with the system default currency (and then use currency rates to calculate the right price).
- **UnitID**
  The ID of the product unit that the system needs to resolve a price for.
- **User**
  The Extranet user that the system is resolving a price for.

2.2 Raw Price

When the system calls a *PriceProvider* for a price it expects the *PriceProvider* to return a *RawPrice* object. A *RawPrice* is a price without a context, consisting only of an amount and a currency. (10 USD or 23,54 DKK). The context, in

 © 2007-2008 Dynamicweb Software A/S (Ltd).

which the price was retrieved from the *PriceProvider*, will be remembered by the system.

2.3 PriceProviders and sales tax

All sales tax (VAT) calculations are done by the system, according to the system configurations.

The only thing a *PriceProvider* needs to check is whether the *PricesInDbWithVAT* bit is set in the configuration. If the bit is true, the *PriceProvider* always returns the price with sales tax. If the bit is false, the PriceProvider always return prices without sales tax.

2.4 Currency and rounding

There are no changes in the way the system deals with prices from a *PriceProvider* regarding currency rates and rounding of prices.

2.5 Code sample

The following code sample shows how to make a *PriceProvider* that subtracts 10% from the default product price for extranet users and 15% of the price if they buy more than five items of the same product.

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
Imports Dynamicweb
Imports Dynamicweb.eCommerce
Imports Dynamicweb.eCommerce.Products
Imports Dynamicweb.eCommerce.Prices

Public Class TestPriceProvider
    Inherits PriceProvider

    Public Overrides Function FindPrice(ByVal Product As Dynamicweb.eCommerce.Products.Product, ByVal Quantity As
Double, ByVal VariantID As String, ByVal Currency As Dynamicweb.eCommerce.International.Currency, ByVal UnitID As
String, ByVal User As Dynamicweb.Frontend.Extranet) As PriceRaw

        ' Get the price from the DefaultPriceProvider
        Dim RawPrice As PriceRaw = New DefaultPriceProvider().FindPrice(Product, Quantity, VariantID, Currency,
UnitID, User)

        ' Check if user is an extranet user
        If User.LoggedIn Then
            If Quantity > 5 Then
                RawPrice.Price *= 0.85
                Return RawPrice
            Else
                RawPrice.Price *= 0.9
                Return RawPrice
            End If
        End If

        ' Continue to the next provider in the provider chain.
        Return Nothing

    End Function

End Class
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

3 FEE PROVIDER

The *FeeProvider* allows you to manipulate the shipping fee of an order in eCommerce using a class that inherits from `Dynamicweb.eCommerce.Orders.FeeProvider`. The class must have a function that overrides the `FindFee` function of the `FeeProvider` base class and returns a `PriceRaw` object.

The following example demonstrates how to grant free shipping to customers who posts multiple orders within 24 hours. If the web shop can ship the items from the first and second order together, there is no need for the customer to be double-charged for shipping and handling.

In cases where no previous orders have been submitted, the *FeeProvider* simply returns null which then will cause Dynamicweb eCommerce to ignore the *FeeProvider* when it sorts out the shipping fee.

```
public class FeeProvider:Dynamicweb.eCommerce.Orders.FeeProvider
{
    public override Dynamicweb.eCommerce.Prices.PriceRaw
FindFee(Dynamicweb.eCommerce.Orders.Order Order)
    {
        Dynamicweb.eCommerce.Prices.PriceRaw ReturnFee = null;

        if (Order.CustomerEmail != "")
        {
            IDataReader DR = Database.getDataReader(
               string.Format("SELECT TOP 1 * FROM EcomOrders WHERE
OrderCustomerEmail = '{0}' AND
               OrderDate >= DATEADD(day, -1, GETDATE()) AND
OrderComplete = {1}", Order.CustomerEmail, Database.SqlBool(true)),
               "Ecom.mdb");
            if (DR.Read())
            {
                ReturnFee = new
Dynamicweb.eCommerce.Prices.PriceRaw(0.00,
Dynamicweb.eCommerce.Common.Application.DefaultCurrency);
            }
        }
        return ReturnFee;
    }
}
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

4 SALES DISCOUNT PROVIDERS

A sales discount defines a set of conditions under which a customer should receive a discount on his purchase if he succeeds meeting certain conditions.

The two standard conditions for all Sales discounts are:

- Is this discount active?
- Is this discount valid for this customer?

It is up to the shop-administrator to configure the discounts and who it should be valid for. There are four user restriction levels:

- Everybody
- All authenticated users
- One specified user or a user group
- Multiple users and user groups

Added to these conditions is the name of the discount, for example “Christmas 2007”.

4.1 Different types of sales discounts

Sales discounts come in different types, depending on the conditions they need to meet. It can be a Christmas discount where everybody should get 5% discount if they buy products for more than 1000 kr. in one order in December. Or it can be a total item discount where everybody who buys more than 10 products gets a free keychain.

4.2 Sales discount types and their parameters

Every sales discount type has its own set of parameters that are used to define the conditions that activates the discount. You can decide the parameter type e.g. a date value, a numerical value or any other type of value.

4.3 Creating your own sales discount

What if none of the standard sales discount types meet the requirements of my webshop? In that case you need to use the *Extensibility API*. It provides you with the possibility to make your own type of discounts.

4.4 How are the SalesDiscounts evaluated?

There is a *SalesDiscountProvider* class that gives the structure that every *SalesDiscountAddIn* should have. All *SalesDiscountAddIns* need to override a function called `ProcessOrder`, which is the function that is called when Dynamicweb eCommerce goes through all the *SalesDiscountAddIns* searching for discounts. This allows the whole procedure to be more flexible, allowing you to decide how your *SalesDiscountAddIn* should work. Details about how this is done can be seen in the code sample below.

4.5 Code sample

This code sample shows how to make a *SalesDiscountProvider* that triggers a discount if the total sum of purchased products (“Order total amount”) exceeds the specified amount.

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
Imports Dynamicweb.eCommerce
Imports Dynamicweb.eCommerce.Prices
Imports Dynamicweb.eCommerce.Orders
Imports Dynamicweb.eCommerce.Products
Imports Dynamicweb.Extensibility

Namespace eCommerce.Orders.SalesDiscounts

    < _
    AddInName("Total price discount"), _
    AddInDescription("If the total sum of the products purchased on an order is equal or bigger that the
specified amount a discount will be triggered.") _
    > _
    Public Class TotalSalesPriceDiscount
        Inherits SalesDiscountProvider

#Region "Fields"

        Private _MoreLess As String = ""
        Private _BuyLimit As Double = 1000.0
        Private _DiscountValue As New DiscountValueHandler

#End Region

#Region "Properties"

        <AddInParameter("Condition"), AddInParameterEditor(GetType(Editors.MoreLessParameterEditor), "")> _
        Public Property MoreLess() As String
            Get
                Return _MoreLess
            End Get
            Set(ByVal Value As String)
                _MoreLess = Value
            End Set
        End Property

        <AddInParameter("Order buylimit"), AddInParameterEditor(GetType(Editors.TextParameterEditor), "")> _
        Public Property BuyLimit() As Double
            Get
                Return _BuyLimit
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
            End Get
            Set(ByVal Value As Double)
                _BuyLimit = Value
            End Set
        End Property

        <AddInParameter("Discount"), AddInParameterEditor(GetType(Editors.DiscountValueEditor), "")> _
        Public Property DiscountValue() As String
            Get
                If _DiscountValue Is Nothing Then
                    _DiscountValue = New DiscountValueHandler()
                End If
                Return _DiscountValue.Value
            End Get
            Set(ByVal Value As String)
                If _DiscountValue Is Nothing Then
                    _DiscountValue = New DiscountValueHandler()
                End If
                _DiscountValue.Value = Value
            End Set
        End Property

#End Region

#Region "Methods"

        Public Overrides Sub ProcessOrder(ByVal _order As Order)

            Dim AmountValid As Boolean = False

            If Me.MoreLess = ">" AndAlso _order.PriceBeforeFees.Price > Me.BuyLimit Then
                AmountValid = True
            ElseIf Me.MoreLess = ">=" AndAlso _order.PriceBeforeFees.Price >= Me.BuyLimit Then
                AmountValid = True
            ElseIf Me.MoreLess = "==" AndAlso _order.PriceBeforeFees.Price = Me.BuyLimit Then
                AmountValid = True
            ElseIf Me.MoreLess = "<" AndAlso _order.PriceBeforeFees.Price < Me.BuyLimit Then
                AmountValid = True
            ElseIf Me.MoreLess = "<=" AndAlso _order.PriceBeforeFees.Price <= Me.BuyLimit Then
                AmountValid = True
            End If
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
            If AmountValid = True Then

                If Me._DiscountValue.Type = DiscountTypes.Product Or Me._DiscountValue.Type =
DiscountTypes.Products Then

                    For Each product As Products.Product In Me._DiscountValue.Products

                        Dim line As New OrderLine
                        line.OrderID = _order.ID
                        line.Order = _order
                        line.ProductID = product.ID
                        line.ProductVariantID = product.VariantID
                        line.Quantity = 1
                        line.Modified = Now
                        line.ProductName = product.Name
                        line.ProductNumber = product.Number
                        line.Reference = "Default.aspx?ID=" & PageId & "&ProductID=" & product.ID & "&VariantID="
& product.VariantID

                        line.Type = Base.ChkString(Base.ChkNumber(OrderLine.OrderLineType.Discount))

                        line.PageID = PageId

                        line.Product = product
                        _order.OrderLines.Add(line, False)

                    Next

                Else
                    ' Calculate discount
                    Dim discountPrice As Double

                    ' How are we going to calculate this?
                    If Me._DiscountValue.Type = DiscountTypes.Percent Then
                        discountPrice = (_order.PriceBeforeFees.Price / 100) * Me._DiscountValue.Amount
                    ElseIf Me._DiscountValue.Type = DiscountTypes.FixedAmount Then
                        discountPrice = Me._DiscountValue.Amount
                    End If

                    ' Get rawprice of the discount.
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
                    Dim rawprice As New PriceRaw(discountPrice,
Dynamicweb.eCommerce.Common.Application.DefaultCurrency)
```

```
                    ' Convert to a calculated price
                    Dim CalcPrice As New Prices.PriceCalculated(rawprice)
                    CalcPrice.PriceWithoutVAT = CalcPrice.PriceWithVAT
                    CalcPrice.VATPercent = 0
                    CalcPrice.VAT = 0

                    ' We need a negative value
                    discountPrice = CalcPrice.Price - (CalcPrice.Price * 2)

                    ' Add a new order line
                    Dim line As New OrderLine
                    line.Order = _order
                    line.Quantity = 1
                    line.ProductName = Me.DiscountName
                    line.SetUnitPrice(discountPrice)
                    line.Type = Base.ChkString(Base.ChkNumber(OrderLine.OrderLineType.Discount))

                    ' Insert orderline
                    _order.OrderLines.Add(line, False)

                End If
            End If

        End Sub

#End Region

    End Class

End Namespace
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

5 GATEWAY PROVIDERS

The *GatewayProvider* allows you to render different template tags, like a MD5 key that is used in most cases but are encrypted differently for each gateway partner.

Current online gateway payment partners in Dynamicweb eCommerce include:

- DebiTech
- DIBS
- ePay
- PayPal
- PBS
- QuickPay
- SecPay

5.1 Code sample

This *GatewayProvider* provides a textfield editor, and renders the value as a tag that can be used in the form template for this gateway.

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
Imports Dynamicweb.Extensibility

Namespace eCommerce.Orders.Gateways

   < _
   AddInName("MyGatewayName"), _
   AddInDescription("MyGatewayDescription") _
   > _
   Public Class PayPal
      Inherits GatewayProvider

      Private _MyParameter As String

      Public Overrides Sub Render(ByVal template As Templatev2.Template,
ByVal TagPreFix As String)
         MyBase.Render(template, TagPreFix)

         template.SetTag(TagPreFix & ".MyParameter", MyParameter)
      End Sub


      <AddInParameter("MyParameter "),
AddInParameterEditor(GetType(Dynamicweb.Extensibility.TextParameterEditor),"")> _
      Public Property MyParameter () As String
         Get
            Return _MyParameter
         End Get
         Set(ByVal Value As String)
          _MyParameter = Value
         End Set
      End Property
   End Class

End Namespace
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

6 STATISTICS PROVIDERS

The *StatisticsProvider* allows you to create your own statistic chart-type that is not included in the standard types in the Statistics, eCom module.

The Statistics,eCom module is integrated with the third party component, *FusionCharts* (http://www.fusioncharts.com)

6.1 Delivering data

The statistics takes in a *DataTable* and shows the data in the graph and list.

6.2 Extra information

By overriding the *Info* property you can add additional information to the statistic view. This property returns a *Hashtable*.

6.3 Own XML data format

If you want to generate your own XML data according to the *FusionCharts* documentation, you can override the *XMLData* property in your *StatisticsProvider*.

6.4 Change the default graph look

By overriding the *DefaultGraph* property, you can change the look of the graph. There are 3 graph-types available:

- Line
- Pie
- Column

6.5 Code sample

The following code sample shows how to make a StatisticsProvider. The sample below is from the *Currency* statistics (standard in the Statistics, eCom module):

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
Imports Dynamicweb.Extensibility
Imports Dynamicweb.Extensibility.Editors

Namespace eCommerce.Statistics

    < _
    AddInName("Currency"), _
    AddInDescription("Shows the currency total in a specified period"), _
    AddInGroup("Orders"), _
    AddInImage("tree/btn_currency.png") _
    > _
    Public Class Currency
        Inherits StatisticsProvider
        Implements IDropDownOptions

#Region "Fields"

        Private _DateFrom As System.DateTime = System.DateTime.Now.AddMonths(-1)
        Private _DateTo As System.DateTime = System.DateTime.Now

        Private _Shop As String = ""
        Private _statDataTable As DataTable = Nothing

#End Region

#Region "Properties"

        <AddInParameter("Date from"), AddInParameterEditor(GetType(Editors.DateTimeParameterEditor), "")> _
        Public Property DateFrom() As System.DateTime
            Get
                Return _DateFrom
            End Get
            Set(ByVal Value As System.DateTime)
                _DateFrom = Value
            End Set
        End Property

        <AddInParameter("Date to"), AddInParameterEditor(GetType(Editors.DateTimeParameterEditor), "")> _
        Public Property DateTo() As System.DateTime
            Get
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
                Return _DateTo
            End Get
            Set(ByVal Value As System.DateTime)
                _DateTo = Value
            End Set
        End Property

        <AddInParameter("Shop"), AddInParameterEditor(GetType(Editors.DropDownParameterEditor), "nonetext=Alle")>
_
        Public Property Shop() As String
            Get
                Return _Shop
            End Get
            Set(ByVal Value As String)
                _Shop = Value
            End Set
        End Property

        Public Function GetOptions(ByVal Name As String) As System.Collections.Hashtable Implements
IDropDownOptions.GetOptions
            ' Get Hashtable
            Dim htOptions As New Hashtable

            Select Case Name
                Case "Shop"
                    Dim shops As eCommerce.Shops.ShopCollection = eCommerce.Common.Application.Shops
                    For Each s As eCommerce.Shops.Shop In shops
                        htOptions.Add(s.ID, s.Name)
                    Next
            End Select

            ' Return the hashtable
            Return htOptions

        End Function

#End Region

        ' Properties
        Public Overrides ReadOnly Property XAxisColumn() As String
            Get
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
            Return "Currency"
        End Get
    End Property

    Public Overrides ReadOnly Property YAxisColumn() As String
        Get
            Return "Orders"
        End Get
    End Property

    Public Overrides ReadOnly Property ColorColunm() As String
        Get
            Return "Color"
        End Get
    End Property

    Public Overrides ReadOnly Property Prefix() As String
        Get
            Return ""
        End Get
    End Property

    Public Overrides ReadOnly Property XMLData() As String
        Get
            Return ""
        End Get
    End Property

    Public Overrides ReadOnly Property DefaultGraph() As GraphType
        Get
            Return GraphType.Column
        End Get
    End Property

    Public Overrides ReadOnly Property Data() As DataTable
        Get
            If _statDataTable Is Nothing Then
                _statDataTable = CollectData()
            End If
            Return _statDataTable
        End Get
    End Property
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
        Private Function CollectData() As DataTable
            Dim myDataTable As DataTable = New DataTable("eCom_Statistic_Order")
            myDataTable.Columns.Add("Color", GetType(String))
            myDataTable.Columns.Add("Currency", GetType(String))
            myDataTable.Columns.Add("Orders", GetType(Double))

            Dim SQL As String = "SELECT EcomOrders.OrderCurrencyCode, EcomCurrencies.CurrencyName, COUNT(*) AS
OrderDateCounter "
            SQL += "FROM EcomOrders "
            SQL += "INNER JOIN EcomOrderStates ON EcomOrderStates.OrderStateID = EcomOrders.OrderStateID "
            SQL += "LEFT JOIN EcomCurrencies ON EcomCurrencies.CurrencyCode = EcomOrders.OrderCurrencyCode "
            SQL += "WHERE EcomOrderStates.OrderStateDontUseInstatistics = " & Database.SqlBool(False) & " "
            SQL += "AND EcomOrders.OrderComplete = " & Database.SqlBool(True) & " "
            SQL += "AND EcomOrders.OrderDeleted = " & Database.SqlBool(False) & " "
            SQL += "AND EcomOrders.OrderDate Between " & Database.SqlDate(DateFrom) & " AND " &
Database.SqlDate(DateTo) & " "
            If Shop <> "" Then
                SQL += "AND EcomOrders.OrderShopID = '" & Shop & "' "
            End If
            SQL += "GROUP BY EcomOrders.OrderCurrencyCode, EcomCurrencies.CurrencyName "

            'Base.w(SQL)

            Dim objDataReader As IDataReader = Database.getDataReader(SQL, "Ecom.mdb")

            Dim cur As String = eCommerce.Common.Application.DefaultCurrency.Code
            Do While objDataReader.Read()

                If Base.ChkString(objDataReader("OrderCurrencyCode")) = "" Then
                    cur = eCommerce.Common.Application.DefaultCurrency.Code
                Else
                    cur = Base.ChkString(objDataReader("OrderCurrencyCode"))
                End If

                Dim myDataRow As DataRow = myDataTable.NewRow()
                myDataRow("Color") = "c00000"
                myDataRow("Currency") = cur
                myDataRow("Orders") = objDataReader("OrderDateCounter")
                myDataTable.Rows.Add(myDataRow)
            Loop
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
            objDataReader.Close()
            objDataReader.Dispose()

            Return myDataTable
        End Function

    End Class

End Namespace
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

7 IMPORT/EXPORT ACTIVITY PROVIDERS

The *Import/Export* module uses so-called pipelines to move data in and out of Dynamicweb. The way data is moved depends on how you configure the pipeline. Pipelines are configured by choosing activities and placing them in a sequence. The module stores all pipelines in an XML file format in the Dynamicweb *File Manager*.

An activity in detail

An activity is like a link in a big chain. The first link in the chain retrieves data and gives it to the next link, and that link gives it to the next and so on until the end of the chain is reached. This procedure is done over and over again until the first link has no more data.

Through the *Extensibility API* you can create properties for activities that are treated as settings for the activity when designing pipelines. By right-clicking on an activity you can see a window that contains a form that is rendered according to the set properties. This reduces the work considerably because no UI coding is needed, and you can concentrate on the functional code.

7.1 Datatype conditions.

An activity defines what type of data it inputs and what type of data it outputs. It does this through an *Attribute* called `AddInDataConstrain` that then contains the data conditions.

- `"*","XML"` would for instance mean that it can input anything but always outputs XML.
- `"","XML"` would mean that it has no input but returns XML.

What are the limitations?

The limits are your imagination. You can retrieve or send data from a local file, a remote server, a database, a web service or other data source/destinations.

7.2 What about reports?

In the beta release only “on-screen” reporting is available, later in the stable version there will follow a possibility of receiving reports by e-mail or by saving them to files. Both should be very useful in combination with scheduled tasks.

7.3 How do I decide what my activity should do?

You do not need to load any settings as the *Extensibility API* should already have provided the activity properties with its correct values according to the settings given when designing the pipeline.

To achieve correct functionality there are several functions and properties that you must override:

1. The `Open` function that is run only when the activity is initialized first and allows you to do some preparation, like opening database connections or open a stream reader to a text file.

 © 2007-2008 Dynamicweb Software A/S (Ltd).

2. The `HasData` property that tells the next activity if your activity has more data to contribute to the pipeline.
3. The `Process` function which is run if the `HasData` returns true. This function would then contain the main code of your activity.
4. The `Close` function which is invoked as soon as the first activity in the pipeline has no more data.

These four are what most activities need to override in order to function correct. The `Open` and `Close` functions are optional and only needed in cases where something has to be invoked or disposed.

7.4 Code sample - Activity

This code sample shows how to make your own activity which you can use in the configuration of pipelines. The sample activity reads a text file line by line.

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
Imports System
Imports System.IO
Imports System.Collections.Generic
Imports System.Text

Imports Dynamicweb.Extensibility
Imports Dynamicweb.Extensibility.Editors

Namespace Integration

    < _
    AddInName("Textfile Reader"), _
    AddInGroup("Files"), _
    AddInImage("image.png"), _
    AddInDescription("Reads out of a textfile line by line."), _
    AddInActive(False), _
    AddInDataConstrain("", "XML") _
    > _
    Public Class TextFileReaderActivity
        Inherits Activity


        Private reader As StreamReader

        Private _filename As String = ""

        <
        AddInParameter("File"), _
        AddInParameterEditor(GetType(Editors.FileSelectEditor), "") _
        > _
        Public Property Filename() As String
            Get
                Return _filename
            End Get
            Set(ByVal Value As String)
                _filename = Value
            End Set
        End Property

        Public Overrides ReadOnly Property HasData() As Boolean
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
            Get
                Return Not reader.EndOfStream
            End Get
        End Property

        Public Overrides Sub Open()
            Dim path As String = HttpContext.Current.Server.MapPath("\Files") & Filename
            Dim fi As New FileInfo(path)
            If fi.Exists Then
                reader = New StreamReader(Filename)
            Else
                Throw New Exception("File not found")
            End If
        End Sub

        Public Overrides Sub Close()
            reader.Close()
            reader.Dispose()
        End Sub

        Public Overrides Sub Process()
            Output = reader.ReadLine()
        End Sub

    End Class

End Namespace
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

8 NOTIFICATION SUBSCRIBER

8.1 Notifications

There are multiple events you can choose from to interact with, and you can also make your own events in your own custom modules. The standard Dynamicweb notifications are part of the *Dynamicweb.Notifications* namespace.

How do I choose what to react to?

Simply by using the *SubscribeAttribute* on your own *NotificationSubcriber* class. You can also have multiple subscriptions on the same class.

Example:

```
<Subscribe(Notifications.Standard.Page.Loaded)>
```

Various subscriber types:

- Standard.Page.Loaded
- Standard.Page.PageTitle
- Standard.User.OnExtranetLogin
- Standard.User.OnExtranetLoginFailed
- eCommerce.Order.State.Changed
- eCommerce.Order.Steps.Completed
- eCommerce.Order.Steps.Confimed
- eCommerce.Order.Steps.CustomerInfoHarvested
- eCommerce.Order.Steps.FeesHarvested
- eCommerce.Order.Steps.PaymentCancelled
- eCommerce.Order.Steps.PaymentFailed
- eCommerce.Order.Steps.PaymentSucceded
- eCommerce.Stock.StatusChanged

How does the API know what events I want?

When an event is triggered, the *NotificationManager* looks for subscribers waiting for this event. If it finds a subscriber it invokes the *OnNotify*-function.

How do I trigger an event?

An event is triggered by using the *NotificationManager*. It has a *Notify*-function where you can pass along what event type that is triggered and an object array containing all objects that could be needed to react to the event.

Example:

```
NotificationManager.Notify(Notifications.Standard.Page.Loaded, New
Object() {Me})
```

8.2 Code samples

The following sections describe multiple usage examples of the *Notification Subscriber*.

 © 2007-2008 Dynamicweb Software A/S (Ltd).

8.2.1 Basic example

This code sample shows how to make a *NotificationSubscriber* for system event notifications.

```
Imports Dynamicweb
Imports Dynamicweb.Frontend
Imports Dynamicweb.Extensibility

Namespace eCommerce.Frontend

<Subscribe(Notification.Standard.Page.PageTitle)> _
Public Class PageTitleSubscriber
   Inherits Extensibility.NotificationSubscriber

   Public Overrides Sub OnNotify(ByVal notification As String, ByVal
args As Object())

      ‘Lets get an instance of the pageview
      Dim pv As PageView = CType(args(0), PageView)
      ‘ Now lets add to the current title
      pv.Meta.Title += “ - My own title.”
   End Sub

End Class

End Namespace
```

8.2.2 Product List Modifier

The *Product List Modifier* is an example on a very useful *Notification Subscriber*. In order to create a modifier you need to write a class which inherits from the class `Dynamicweb.Extensibility.NotificationSubscriber` and has the attribute *Subscribe*. The following shows an example where the class subscribes to `Dynamicweb.Notifications.eCommerce.Cart.BeforeRender`, which can be used to change the shopping cart content before it is rendered.

```
    /*
     * Postfix all product names with " - modified!"
     */
    [Subscribe(Dynamicweb.Notifications.eCommerce.Cart.BeforeRender)]
    public class TestCartModifier : NotificationSubscriber
    {
        public override void OnNotify(string notification,
NotificationArgs argsBase)
        {
            Dynamicweb.Notifications.eCommerce.Cart.BeforeRenderArgs
args =
(Dynamicweb.Notifications.eCommerce.Cart.BeforeRenderArgs)argsBase;
            string postfix = " - modified!";
            foreach (OrderLine orderLine in args.OrderLines)
                if (!orderLine.ProductName.EndsWith(postfix))
                    orderLine.ProductName += postfix;
        }
    }
```

You need to be aware of the following elements when creating your own class:

1. **`[Subscribe(Dynamicweb.Notifications.eCommerce.Cart.BeforeRender)]`**
   Subscribes to the particular notification. In this case a notification will be created every time the cart is rendered.

 © 2007-2008 Dynamicweb Software A/S (Ltd).

2. **public class TestCartModifier : NotificationSubscriber**
   The class needs to inherit from the `NotificationSubscriber` (in the namespace `Dynamicweb.Extensibility`).
3. **public override void OnNotify(string notification, NotificationArgs argsBase)**
   This is the method which is called at every notification.
4. **Dynamicweb.Notifications.eCommerce.Cart.BeforeRenderArgs args = (Dynamicweb.Notifications.eCommerce.Cart.BeforeRenderArgs)argsBase;**
   You can type caste the arguments for this particular notification. In this case `Cart.BeforeRenderArgs` contains an `OrderLineCollection` which can be modified.

 © 2007-2008 Dynamicweb Software A/S (Ltd).

The following section shows three examples (including imports):

```
using Dynamicweb.eCommerce.Products;
using Dynamicweb.eCommerce.Orders;
using Dynamicweb.Extensibility;

namespace ModifierDemo
{

    /*
     * Postfix all product names with " - modified!"
     */
    [Subscribe(Dynamicweb.Notifications.eCommerce.Cart.BeforeRender)]
    public class TestCartModifier : NotificationSubscriber
    {
        public override void OnNotify(string notification, NotificationArgs argsBase)
        {
            Dynamicweb.Notifications.eCommerce.Cart.BeforeRenderArgs args =
(Dynamicweb.Notifications.eCommerce.Cart.BeforeRenderArgs)argsBase;
            string postfix = " - modified!";
            foreach (OrderLine orderLine in args.OrderLines)
                if (!orderLine.ProductName.EndsWith(postfix))
                    orderLine.ProductName += postfix;
        }
    }

    /*
     * Remove all products that cost less than 100
     */
    [Subscribe(Dynamicweb.Notifications.eCommerce.ProductList.BeforeSort)]
    public class TestProductListModifier : NotificationSubscriber
    {
        public override void OnNotify(string notification, NotificationArgs argsBase)
        {
            Dynamicweb.Notifications.eCommerce.ProductList.BeforeSortArgs args =
(Dynamicweb.Notifications.eCommerce.ProductList.BeforeSortArgs)argsBase;
            for (int i = args.Products.Count - 1; i >= 0; i--)
                if (args.Products[i].Price.Price < 100)
                    args.Products.RemoveAt(i);
        }
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
    }

    /*
     * Modify the product name to include a whitespace between each char
     */
    [Subscribe(Dynamicweb.Notifications.eCommerce.Product.BeforeRender)]
    public class TestProductModifier : NotificationSubscriber
    {
        public override void OnNotify(string notification, NotificationArgs argsBase)
        {
            Dynamicweb.Notifications.eCommerce.Product.BeforeRenderArgs args =
(Dynamicweb.Notifications.eCommerce.Product.BeforeRenderArgs)argsBase;
            string newName = "";
            foreach (char c in args.Product.Name.ToCharArray())
                newName += c + " ";
            args.Product.Name = newName;
        }
    }

}
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

8.2.3 Subscribe to module events

Another example on how to use the *Notification Subscriber* is to subscribe to events when a module is being added or removed from Dynamicweb CMS. The main idea for this *Notification Subscriber* is to take action when a module is being added to the solution. For instance if your custom module has a connection to an external system and this system has a service running that is causing a heavy load, you might consider closing this service when your custom module if being removed. Another scenario where this *Notification Subscriber* comes in handy is for keeping track of which modules your customers have installed. This can provide your financial department with valuable information for sending out invoices related to module purchases.

Notice that you should use this functionality with caution as it allows you to log all traffic regarding module enabling/disabling on the solution.

```
using System;
using System.Collections.Generic;
using System.Text;

using Dynamicweb;
using Dynamicweb.Extensibility;

namespace ModuleEvents
{
    [Subscribe(Dynamicweb.Notifications.Module.Added)]
    public class MyModuleAddedNotificationSubscriber :
NotificationSubscriber
    {
        public override void OnNotify(string notification, object[]
args)
        {
            Dynamicweb.Backend.Module.Module m =
(Dynamicweb.Backend.Module.Module)args[0];
            if (m.SystemName == "MyModuleSystemName")
            {
                //.............................
            }
        }
    }
    [Subscribe(Dynamicweb.Notifications.Module.Removed)]
    public class MyModuleRemovedNotificationSubscriber :
NotificationSubscriber
    {
        public override void OnNotify(string notification, object[]
args)
        {
            Dynamicweb.Backend.Module.Module m =
(Dynamicweb.Backend.Module.Module)args[0];
            if (m.SystemName == "MyModuleSystemName")
            {
                //.............................
            }
        }
    }
}
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

8.2.4 Extranet login notification

Another great way to use the Notification Subscriber is to create notifications when extranet users log on to extranet pages. In that case you will need to subscribe to the following notifications:

- `Dynamicweb.Notifications.Standard.User.OnExtranetLogin`
- `Dynamicweb.Notifications.Standard.User.OnExtranetLoginFailed`

You can also add arguments to the notifications allowing you to display messages to users when they are logging in. The following shows an example of two classes that subscribes to the two extranet notifications:

```
    [Subscribe(Dynamicweb.Notifications.Standard.User.OnExtranetLogin)]
    public class LoginSucceeded : NotificationSubscriber
    {
        public override void OnNotify(string notification,
NotificationArgs args)
        {
            Dynamicweb.Notifications.Standard.User.OnExtranetLoginArgs
MyArgs =
(Dynamicweb.Notifications.Standard.User.OnExtranetLoginArgs)args;
            Dynamicweb.Base.wa("Welcome " + MyArgs.User.Name);
        }
    }


[Subscribe(Dynamicweb.Notifications.Standard.User.OnExtranetLoginFailed)
]
    public class LoginFailed : NotificationSubscriber
    {
        public override void OnNotify(string notification,
NotificationArgs args)
        {

Dynamicweb.Notifications.Standard.User.OnExtranetLoginFailedArgs MyArgs
=
(Dynamicweb.Notifications.Standard.User.OnExtranetLoginFailedArgs)args;
            switch (MyArgs.Reason)
            {
                case
Dynamicweb.Notifications.Standard.User.OnExtranetLoginFailedArgs.FailRea
son.IncorrectLogin: Dynamicweb.Base.wa("The combination of user name and
password does not exist."); break;
                case
Dynamicweb.Notifications.Standard.User.OnExtranetLoginFailedArgs.FailRea
son.PasswordExpired: Dynamicweb.Base.wa("Your password has expired.
Please contact webmaster"); break;
                case
Dynamicweb.Notifications.Standard.User.OnExtranetLoginFailedArgs.FailRea
son.PasswordLengthInvalid: Dynamicweb.Base.wa("Password length is
invalid."); break;

            }
        }
    }
```

Notice that the `OnExtranetLoginFailed` arguments have a reason property telling what went wrong when the user tried to log in.

8.2.5 Custom order line fields modifier

The final example shows how you can manipulate the information in custom order line fields. The following shows an example where a class subscribes to

 © 2007-2008 Dynamicweb Software A/S (Ltd).

the notification `Dynamicweb.Notifications.eCommerce.Order.Steps.Completed`, which can be used to alter the custom order line field information.

```
[Subscribe(Dynamicweb.Notifications.eCommerce.Order.Steps.Completed)]
    public class NotificationSubscriber2 : NotificationSubscriber
    {

        public override void OnNotify(string notification, object[]
args)
        {
            Dynamicweb.eCommerce.Orders.Order MyOrder =
(Dynamicweb.eCommerce.Orders.Order)args[0];
            foreach (Dynamicweb.eCommerce.Orders.OrderLine line in
MyOrder.OrderLines)
            {
                foreach (Dynamicweb.eCommerce.Orders.OrderLineFieldValue
field in line.OrderLineFieldValues)
                {
                    if (field.OrderLineFieldSystemName.ToLower() ==
"dispatchestimate")
                    {
                        field.Value = "Week 45";
                        line.Save();
                        break;
                    }
                }
            }
        }
    }
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

9 TEMPLATE EXTENDERS

Template Extenders provide you with a common method for defining new tags and overriding existing tag values in Dynamicweb eCommerce templates from within a custom module or an independent assembly in the eCommerce application domain.

Template Extenders are basically classes that derives from predetermined base classes and hooks into the default rendering process, allowing you to manipulate templates and their associated data in a given process or state.

9.1 Extender types

There are currently five kinds of *Template Extenders*. Additional extenders are planned in future releases:

9.1.1 ProductTemplateExtender

Processed when the system renders a product on a product page or a single product in a product list page. The scope of this *Extender* is the current *Product* object being rendered.

9.1.2 ProductListTemplateExtender

Processed when the system renders a product list. The scope of the *Extender* is the current *ProductCollection* object being rendered.

9.1.3 OrderTemplateExtender

Processed before and after the system renders an *Order* or a *Cart*. The scope is the current *Order* object being rendered.

9.1.4 OrderLineTemplateExtender

Processed before and after the system renders an *OrderLine* or a *CartLine*. The scope is the current *OrderLine* being rendered.

9.1.5 PageTemplateExtender

Processed when the system renders a page. The scope of the *Extender* is the current *Page* object being rendered.

9.2 Template Extenders RenderingState

When processed, some of the template extenders are being called with a rendering state as a parameter. There are currently two rendering states:

- **Before**
  The *Extender* is called before the system creates tags for a template or a template loop. Do not try to override system tags here, as the system will take control later on, and your changes will be overridden by the default rendering mechanism. Instead, use this state to define your own tags or manipulate data objects (*Order*, *Product*), as changes made to these will be caught by the rendering process later and reflected in the template tags.
- **After**
  Use this state to define your own tags and to override system tag values. Because the system has finished the rendering process in this state, your changes to system tags will be kept and reflected in the template.

 © 2007-2008 Dynamicweb Software A/S (Ltd).

Template Extenders are stateless. *Extenders* supporting the rendering state parameter are called in both states, but with different object instances. So do not try to set a variable or property in the “before” state and try retrieving it in the “after” state. Extenders that do not support *RenderingState* are only called in the “After” state.

9.3 Code samples

9.3.1 ProductTemplateExtender

Rendering States: No

Example:

```
Public Class SampleProductTemplateExtender
    Inherits ProductTemplateExtender

    Public Overrides Sub ExtendTemplate(ByVal Template As
Templatev2.Template)
        Template.SetTag("Ecom:Product.TestTag1", "HELLO")
        Template.SetTag("Ecom:Product.TestTag2", "WORLD")
        If Not Order Is Nothing Then
            Template.SetTag("Ecom:Order:Price",
Order.Price.PriceFormatted)
        Else
            Template.SetTag("Ecom:Order:Price", "Nothing in the basket")
        End If
    End Sub

End Class
```

9.3.2 ProductListTemplateExtender

Rendering States: No

Example:

```
Public Class SampleProductListTemplateExtender
    Inherits ProductListTemplateExtender

    Public Overrides Sub ExtendTemplate(ByVal Template As
Templatev2.Template)
        Template.SetTag("Ecom:Product.GimeTheNumsOfProds",
ProductList.Count.ToString() & "Products")
        If Not Order Is Nothing Then
            Template.SetTag("Ecom:Order:Price",
Order.Price.PriceFormatted)
        Else
            Template.SetTag("Ecom:Order:Price", "Nothing in basket")
        End If
    End Sub

End Class
```

9.3.3 OrderTemplateExtender

Rendering States: Yes

Example:

```
Public Class SampleOrderTemplateExtender
    Inherits OrderTemplateExtender
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

```
    Public Overrides Sub ExtendTemplate(ByVal Template As
Templatev2.Template, ByVal RenderingState As
TemplateExtenderRenderingState)
```

```
        If RenderingState = TemplateExtenderRenderingState.Before Then
            Template.SetTag("Ecom:Order.BeforeRenderTag", "BEFORE THE
ORDER WAS RENDERED (" & Order.Price.PriceFormatted & ")")
        Else
            Template.SetTag("Ecom:Order.AfterRenderTag", "AFTER THE
ORDER WAS RENDERED (" & Order.Price.PriceFormatted & ")")
        End If
    End Sub

End Class
```

9.3.4 OrderLineTemplateExtender

Rendering States: Yes

Example:

```
Public Class SampleOrderLineTemplateExtender
    Inherits OrderLineTemplateExtender

    Public Overrides Sub ExtendTemplate(ByVal Template As
Templatev2.Template, ByVal RenderingState As
TemplateExtenderRenderingState)
        If RenderingState = TemplateExtenderRenderingState.Before Then
            Template.SetTag("Ecom:Order.OrderLine.BeforeRenderTag",
"BEFORE THE ORDERLINE WAS RENDERED (" & OrderLine.Quantity & " x " &
OrderLine.UnitPrice.PriceFormatted & ")")
        Else
            Template.SetTag("Ecom:Order.OrderLine.AfterRenderTag",
"AFTER THE ORDERLINE WAS RENDERED (" & OrderLine.Quantity & " x " &
OrderLine.UnitPrice.PriceFormatted & ")")
        End If
    End Sub

End Class
```

9.3.5 PageTemplateExtender

Rendering States: No

Example:

```
Public Class SamplePageTemplateExtender
    Inherits PageTemplateExtender

    Public Overrides Sub RenderTemplate(ByVal Template As
Templatev2.Template)
        ..............
    End Sub

End Class
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).

10 MENU PROVIDER

The *Menu Provider* allows you to add your own custom menu items in XSLT menus using a class that inherits from `Dynamicweb.Frontend.NavigationProviders`. The class must have a function that overrides the ProcessTree function of the NavigationProvider base class. The function accepts RootNavigationItem as an argument. You can also change existing menu items and add custom attributes to the navigation XML. The following shows an example of a menu provider class:

```
[AddInOrder(-1)]
    public class MenuProvider : NavigationProvider
    {
        public override void
ProcessTree(Dynamicweb.Frontend.NavigationProviders.RootNavigationItem
rootNode, Dynamicweb.Frontend.XMLNavigation.NavigationType
navigationType)
        {

            foreach (NavigationItem n in rootNode)
            {

                if (n.ID == 1)
                {
                    n.MenuText = "Products";

                    MenuItem MyMenuItem = new MenuItem();
                    MyMenuItem.MenuText = "My custom menu item";
                    MyMenuItem.MyCustomAttribute = "My custom value";
                    n.AddChild(MyMenuItem);

                    MenuItem MySubMenuItem = new MenuItem();
                    MySubMenuItem.MenuText = "My custom sub menu item";
                    MySubMenuItem.MyCustomAttribute = "An other custom
value";
                    MyMenuItem.AddChild(MySubMenuItem);
                    break;
                }
            }
            MenuItem m = new MenuItem();
            m.MenuText = "Root level menu item";
            m.MyCustomAttribute = "A menu item in the menu root";
            rootNode.AddChild(m);
        }
    }

    public class MenuItem : NavigationItem
    {
        private string _MyCustomAttribute;


        [AddInName("MyCustomAttribute")]
        public string MyCustomAttribute
        {

            get { return _MyCustomAttribute; }
            set { _MyCustomAttribute = value; }
        }

    }
```

 © 2007-2008 Dynamicweb Software A/S (Ltd).